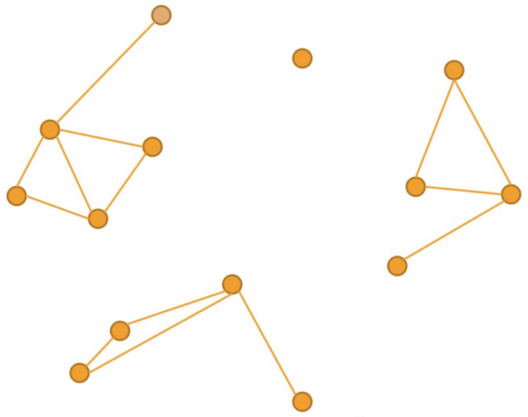
CLEMSON
School of COMPUTING

Union-find data structures

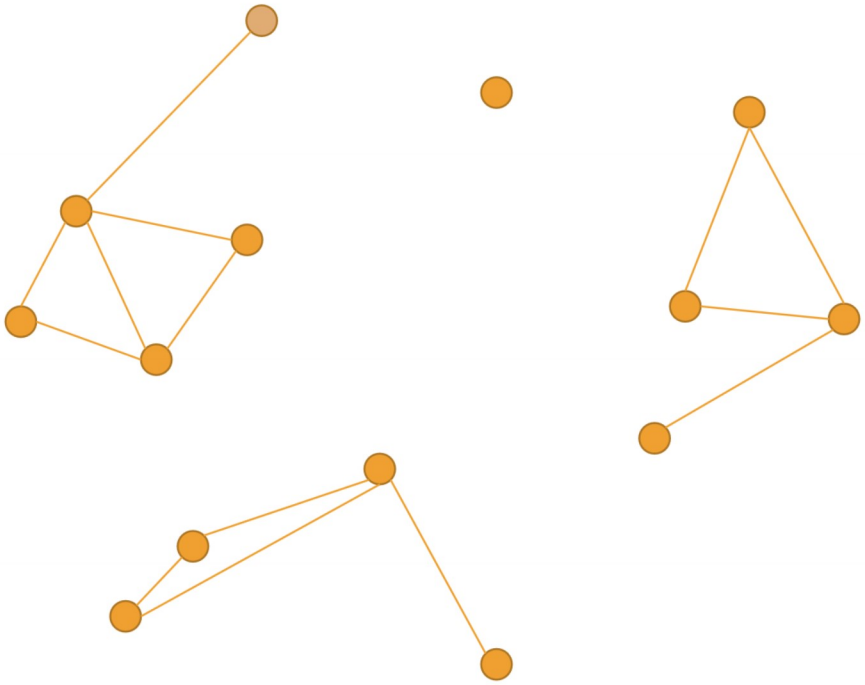
Union Find operations

- Union and find are two operations involved in a number of applications
  - Social networks analysis
  - Containment relationship among segments
  - Percolation theory



How many components do I have in this graph?

Union and Find



How to solve the problem of finding components?

```
function UFConnectedComponents(N,A)
        input – N set of nodes of the graph
        input – A set of arcs of the graph

        for each n in N
                makeSet(n)                          O(N)

        for each (x,y) in A
                if(find(x) != find(y))
                        union(find(x),find(y))      O(A*max(find_c, union_c))

        return number sets
```

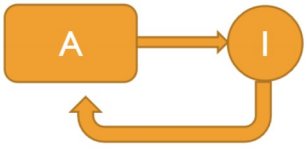
Two efficient implementations

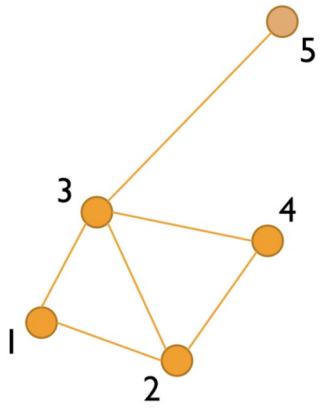
- List-based implementation (our focus today)
- Tree-based implementation (next time)

- A union-find data structure always involves three operations
  - MakeSet(e) – create a singleton with one element e
  - Union(A,B) – generate AUB from A and B
  - Find(e) – return the name of the set containing e

List-based. How does it work?

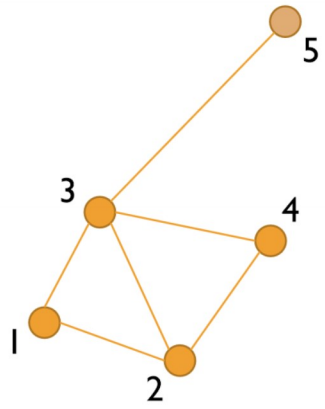
```
Class Header{
        list of elements    //list of elements belonging to the set
        name                //label of the set
}

Class Element{
        head    //pointer to the head of the set
}
```

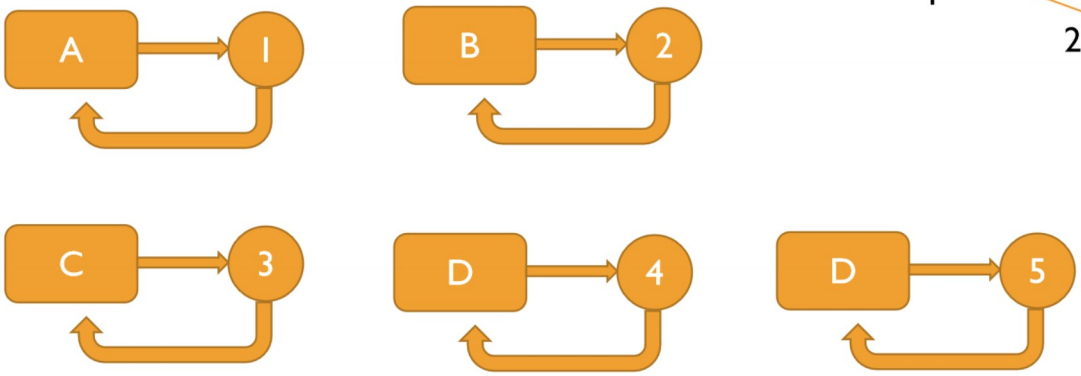




List-based. How does it work?

```
Class Header{
        list of elements    //list of elements belonging to the set
        name                //label of the set
}

Class Element{
        pointer to the head of the set
}
```

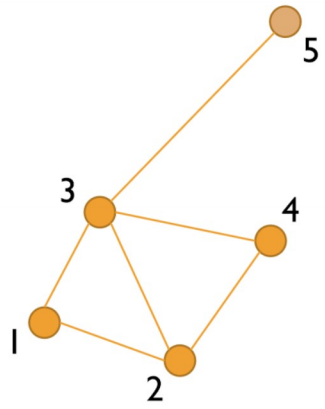


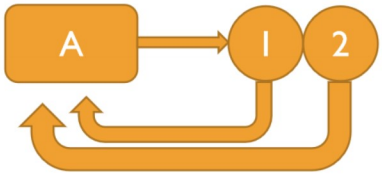


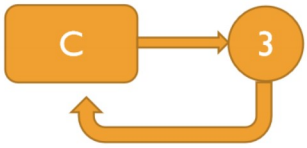
Example of a makeSet on the entire input

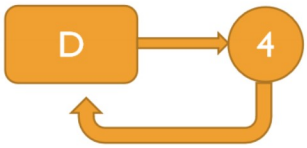
List-based. How does it work?

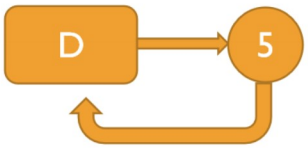
```
Class Header{
        list of elements    //list of elements belonging to the set
        name                //label of the set
}

Class Element{
        pointer to the head of the set
}
```







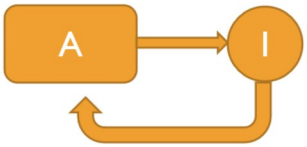




Example of find and unione for (1,2)

List-based. How to implement?

Implement MakeSet

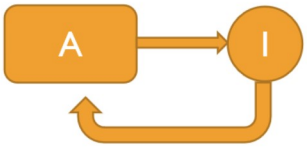



```
function makeSet(e)
        Header u
        u.name = //set a new name
        u.list = //empty list

        Element el = Element(e)
        u.list.push(el)
        el.head = u
```

O(c)

List-based. How to implement?

Implement find

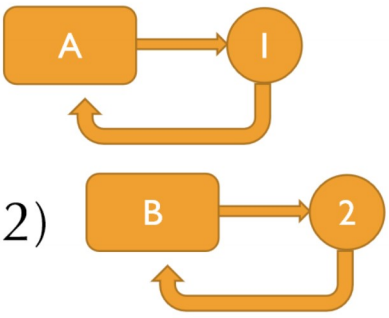



```
function find(Element el)
        return el.head.name
```

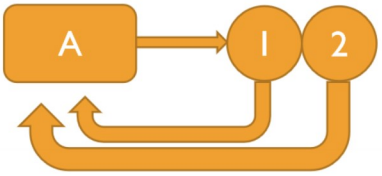
O(c)

List-based. How to implement?

Implement union




```
function union(Header h1, Header h2)
        for each el in h2.list
                h1.list.push(el)
                el.head = h1
```



O(N)

List-based. Evaluation

Single operations

- makeSet O(c)
- find O(c)
- union O(N)

How much is the cost of running a sequence of N of these operations?

$O(N^2)$

Union-by-size

Implement union

```
function union(Header h1, Header h2)
        if(h1.list.size > h2.list.size)
                for each el in h2.list
                        h1.list.push(el)
                        el.head = h1
        else
                for each el in h1.list
                        h2.list.push(el)
                        el.head = h2
```

O(logN)

List-based. Evaluation

Single operations

- makeSet O(c)
- find O(c)
- union O(logN)

How much is the cost of running a sequence of N of these operations?

O(NlogN)

2 steps

4 steps

8 steps

What's next?

- Book chapters
  - Chapter 7.1-7.2
- In class activities
  - Use list-based union-find on new examples
- Next class
  - Union find data structures with tree-based implementation.

List-based Union-Find Structure

Exercise

Show the data structure returned by the FIND-SET operations in the following program. Use the linked-list representation with the weighted-union heuristic.

Assume that if the sets containing x_i and x_j have the same size, then the operation $UNION(x_i, x_j)$ appends x_j's list onto x_i's list.

```
for i = 1 to 16
    MAKE-SET(x[i])
for i = 1 to 15 by 2
    UNION(x[i], x[i + 1])
for i = 1 to 13 by 4
   UNION(x[i], x[i + 2])
UNION(x[1], x[5])
UNION(x[11], x[13])
UNION(x[1], x[10])
FIND-SET(x[2])
FIND-SET(x[9])
```

We use i to represent x_i

```
for i = 1 to 16
  MAKE-SET(x[i])
```
→ {1} {2} {3} {4} {5} {6} {7} {8} {9} {10} {11} {12} {13} {14} {15} {16}

```
for i = 1 to 15 by 2
  UNION(x[i], x[i + 1])
```
→ {1, 2} {3, 4} {5, 6} {7, 8} {9, 10} {11, 12} {13, 14} {15, 16}

```
for i = 1 to 13 by 4
  UNION(x[i], x[i + 2])
```
→ {1, 2, 3, 4} {5, 6, 7, 8} {9, 10, 11, 12} {13, 14, 15, 16}

```
UNION(x[1], x[5])
```
→ {1, 2, 3, 4, 5, 6, 7, 8} {9, 10, 11,12} {13, 14, 15, 16}

```
UNION(x[11], x[13])
```
→ {1, 2, 3, 4, 5, 6, 7, 8} {9, 10, 11,12, 13, 14, 15, 16}

```
UNION(x[1], x[10])
```
→ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,12, 13, 14, 15, 16}

```
FIND-SET(x[2])
FIND-SET(x[9])
```
→ Both return the pointer to x_1